AccuKnox Assessment

Problem Statement 1

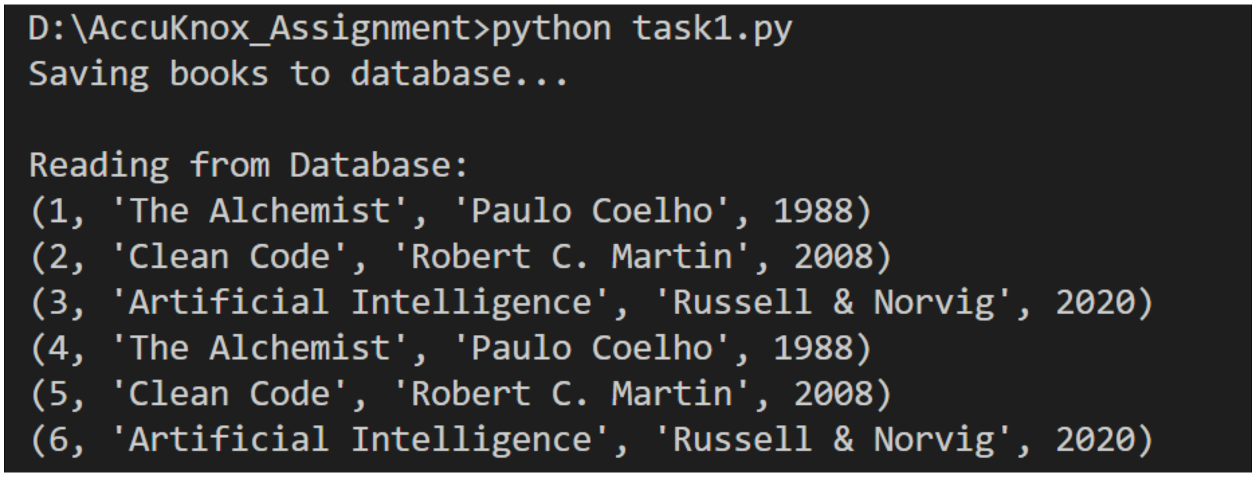
I have implemented the solutions for the three problem statements using Python. The complete source code is hosted on my GitHub repository for review.
GitHub Repository Link: https://github.com/palAdityax08/AccuKnox_Assignment

Task 1: API Data Retrieval and Storage

Objective: Fetch book data from an API, store it in a local SQLite database, and display it. **Approach:**

- Since a live API endpoint was not provided, I mocked the API response within the script (task1_books.py) to simulate the JSON payload.
- I used the sqlite3 library to create a local books.db file and insert the records.
- The script fetches the data back from the database to verify storage was successful.

Execution Output: Below is the terminal output showing the database creation and data retrieval.



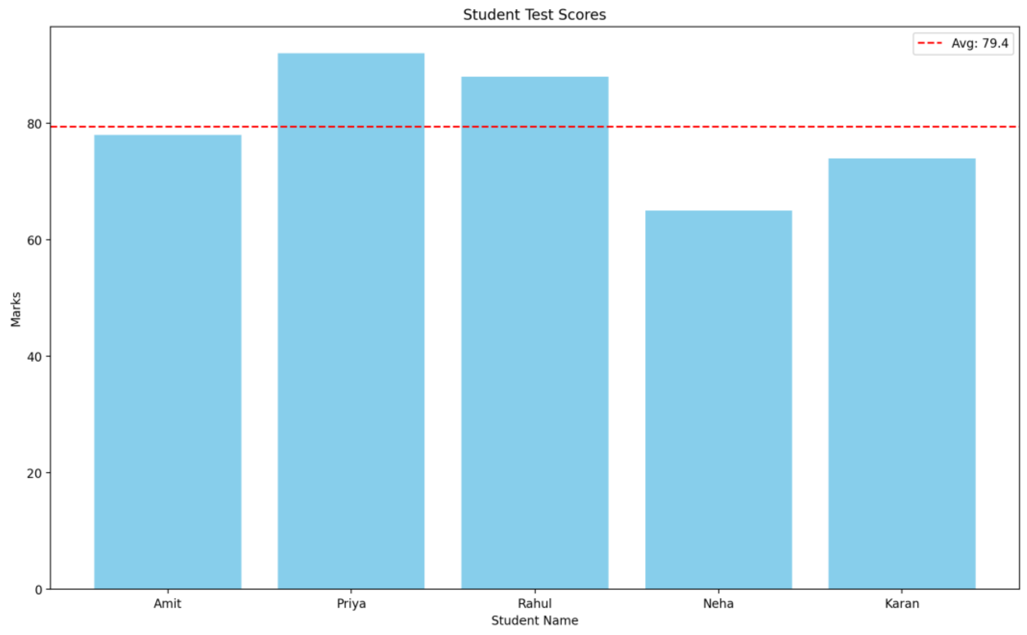
```
D:\AccuKnox_Assignment>python task1.py
Saving books to database...

Reading from Database:
(1, 'The Alchemist', 'Paulo Coelho', 1988)
(2, 'Clean Code', 'Robert C. Martin', 2008)
(3, 'Artificial Intelligence', 'Russell & Norvig', 2020)
(4, 'The Alchemist', 'Paulo Coelho', 1988)
(5, 'Clean Code', 'Robert C. Martin', 2008)
(6, 'Artificial Intelligence', 'Russell & Norvig', 2020)
```

Task 2: Data Processing and Visualization

Objective: Calculate the average score from a dataset and create a visualization.
Approach:

- I used pandas to load the student data and calculate the mean score efficiently.
- I used matplotlib to generate a bar chart. I added a dashed red line to clearly indicate the average score relative to individual student performance.

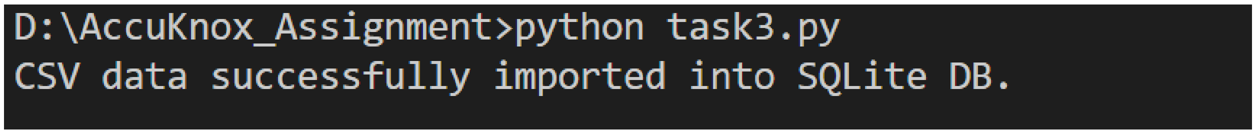
Visualization Output: Below is the generated chart showing student scores.



Task 3: CSV Data Import to Database
Objective: Read user data from a CSV file and insert it into a database.
Approach:

- I created a sample users.csv file containing names and emails.
- The script task3_csv_import.py reads this file using the csv module and inserts the rows into a new users table in SQLite.

Execution Output: Below is the confirmation message after running the import script.



```
D:\AccuKnox_Assignment>python task3.py
CSV data successfully imported into SQLite DB.
```

Problem statement- 2

1. Self-Rating
Rating:

- **A (Independent):** Python, LLMs, AI.
- **B (Supervised):** Deep Learning Theory.

Reasoning: I rated myself an **A** in Python and LLMs because I have independently built end-to-end applications. For example, my recent project **AIKARA** is a fully functional RAG system where I handled the entire pipeline—from PDF ingestion to vector search and Llama 3.2 integration—without requiring supervision.
I rated myself a **B** in Deep Learning because while I can implement models using frameworks like PyTorch or TensorFlow, I am still deepening my understanding of advanced model architectures and mathematical optimization. I am comfortable building, but I value guidance on complex theoretical research.

2. LLM Chatbot Architecture
To create a reliable chatbot that uses Large Language Models (LLMs) without making up facts (hallucinations), I would use a **RAG (Retrieval-Augmented Generation)** architecture.
High-Level Approach: The system does not rely on the LLM's internal memory alone. Instead, it acts like a research assistant:

1. **User asks a question.**
2. **Retrieval:** The system searches a private knowledge base (like company PDFs or database records) to find relevant information.
3. **Generation:** The system sends the user's question *plus* the found information to the LLM and gives the instruction: "Answer the user using only this information."

Key Components:

- **Orchestrator (Backend):** A framework like FastAPI or LangChain to manage the flow.
- **Vector Database:** A specialized database to store text as numbers (embeddings) for fast search.
- **Embedding Model:** A model (like bge-m3) to convert text into vectors.
- **LLM:** The reasoning engine (like GPT-4 or Llama 3) to generate the final response.

3. Vector Databases
What they are: Traditional databases find data by matching exact keywords (e.g., searching "dog" finds "dog"). Vector databases find data by *meaning*. They store data as high-dimensional vectors. This means a search for "canine" will find "dog" because the two words are mathematically close in the vector space.
My Choice for a Hypothetical Problem:

- **Problem:** Building a **Personal Document Search Engine** that runs entirely on a user's laptop (offline).
- **Selected Database: ChromaDB**.
- **Why:** I would choose ChromaDB because it is open-source and can run locally (embedded) within the Python environment. It does not require setting up a separate heavy server (like Docker containers), which makes it perfect for local, privacy-focused applications. It also integrates very easily with LangChain.

Part 3: Complex Code Samples

Most Complex Python Code

- **Project:** AIKARA (AI Teaching Assistant)
- **Link:** https://github.com/palAdityax08/AIKARA
- **Description:** This project implements a complete RAG pipeline. The code handles parsing video transcripts, generating embeddings using bge-m3, and implementing a custom context injection logic to ensure Llama 3.2 provides verifiable, citation-backed answers.

Most Complex Database Code

- **Project:** AccuKnox Assessment Solutions (Task 1 & Task 3)
- **Description:** Since my previous IoT work was on private local hardware (SecureNest), I have demonstrated my database skills in this assessment's repository. Specifically, task1_books.py and task3_csv_import.py showcase my ability to programmatically manage **SQLite** databases. This includes designing schemas, handling bulk data insertion from CSVs, and executing raw SQL queries (CREATE, INSERT, SELECT) directly from Python.